Tree Assignment 4 done by N S K K K Naga Jayanth

https://leetcode.com/problems/recover-binary-search-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    TreeNode a;
    TreeNode b;
    TreeNode prev = new TreeNode(Integer.MIN_VALUE);

    void inorder(TreeNode root) {
        if (root == null)
            return;
        inorder(root.left);
        if (root.val < prev.val && a == null) {
            a = prev;
            b = root;
        } else if (root.val < prev.val && a != null) {
            b = root;
```

```
        }
        prev = root;
        inorder(root.right);
    }

    public void recoverTree(TreeNode root) {
        inorder(root);
        int temp = a.val;
        a.val = b.val;
        b.val = temp;
    }
}
```

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(root == null || root == p || root == q) return root;

        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);

        if(left == null) return right;
        else if(right == null) return left;
        else return root;
    }
}
```

https://leetcode.com/problems/diameter-of-binary-tree/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
```

```
*     TreeNode right;
*     TreeNode() {}
*     TreeNode(int val) { this.val = val; }
*     TreeNode(int val, TreeNode left, TreeNode right) {
*         this.val = val;
*         this.left = left;
*         this.right = right;
*     }
* }
*/
class Solution {
    int maxD=0;
    public int diameterOfBinaryTree(TreeNode root) {
        checkBal(root);
        return(maxD);
    }
    public int checkBal(TreeNode root)
    {
        if(root==null)
            return 0;

        int leftl=checkBal(root.left);
        int rightl=checkBal(root.right);
        maxD=Math.max(maxD,rightl+leftl);
        return (Math.max(leftl,rightl)+1);
    }
}
```

https://practice.geeksforgeeks.org/problems/print-common-nodes-in-bst/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

```
// Definition for a binary tree node.
```

```
class Node {
    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class Solution {
    //Function to find the nodes that are common in both BSTs.
    public static ArrayList<Integer> findCommon(Node root1, Node root2) {
        ArrayList<Integer> result = new ArrayList<>();
        Stack<Node> stack1 = new Stack<>();
        Stack<Node> stack2 = new Stack<>();

        // Helper function to push leftmost nodes to the stack
        // for both BSTs
        populateStack(stack1, root1);
        populateStack(stack2, root2);

        // Compare nodes while both stacks are not empty
        while (!stack1.isEmpty() && !stack2.isEmpty()) {
            Node node1 = stack1.peek();
            Node node2 = stack2.peek();

            if (node1.data == node2.data) {
                // Common node found, add it to the result list
                result.add(node1.data);
                stack1.pop();
```

```
            stack2.pop();
            // Move to the next nodes in both BSTs
            populateStack(stack1, node1.right);
            populateStack(stack2, node2.right);
        } else if (node1.data < node2.data) {
            // If node1 is smaller, move to the next node in BST1
            stack1.pop();
            populateStack(stack1, node1.right);
        } else {
            // If node2 is smaller, move to the next node in BST2
            stack2.pop();
            populateStack(stack2, node2.right);
        }
      }

      return result;
   }

   // Helper function to push leftmost nodes to the stack
   private static void populateStack(Stack<Node> stack, Node root) {
      while (root != null) {
         stack.push(root);
         root = root.left;
      }
   }
}
```

https://leetcode.com/problems/same-tree/

```
class Solution {
   public boolean isSameTree(TreeNode p, TreeNode q) {
      if(p==null && q == null){
         return true;
```

```
        }
        if(p==null || q==null){
            return false;
        }
        if(p.val==q.val){
            return isSameTree(p.left,q.left) && isSameTree(p.right,q.right);

        }
        return false;
    }
}
```

https://leetcode.com/problems/kth-smallest-element-in-a-bst/

```
class Solution {
    public void inorder(TreeNode root,ArrayList<Integer> list)
    {
        if(root==null)
        {
            return ;
        }
        inorder(root.left,list);
        list.add(root.val);
        inorder(root.right,list);
    }
    public int kthSmallest(TreeNode root, int k) {
        ArrayList<Integer> list=new ArrayList<>();
        inorder(root,list);
        return list.get(k-1);
    }
}
```

https://www.interviewbit.com/problems/path-to-given-node/

```
public class Solution {
```

```
    public ArrayList<Integer> solve(TreeNode A, int B) {
        ArrayList<Integer> path = new ArrayList<>();
        findPath(A, B, path);
        return path;
    }

    private boolean findPath(TreeNode node, int target, ArrayList<Integer> path) {
        // Base case: if the node is null, return false
        if (node == null) {
            return false;
        }

        // Add the current node's value to the path
        path.add(node.val);

        // If the current node is the target, return true
        if (node.val == target) {
            return true;
        }

        // Recur for the left and right subtrees
        if (findPath(node.left, target, path) || findPath(node.right, target, path)) {
            return true;
        }

        // If the target node is not found in the current subtree, remove the node from the path
        path.remove(path.size() - 1);
        return false;
    }
}
```

https://leetcode.com/problems/validate-binary-search-tree/

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return check(root,Long.MIN_VALUE,Long.MAX_VALUE);
    }
    public boolean check(TreeNode root, long min, long max){

        if(root==null){
            return true;
        }
        if(root.val<=min || root.val>=max){
            return false;
        }
        return check(root.left, min, root.val) && check(root.right, root.val, max);
    }
}
```